Interactive Minecraft

Alexis Camacho, Chenkai Mao(Kevin)

Motivation

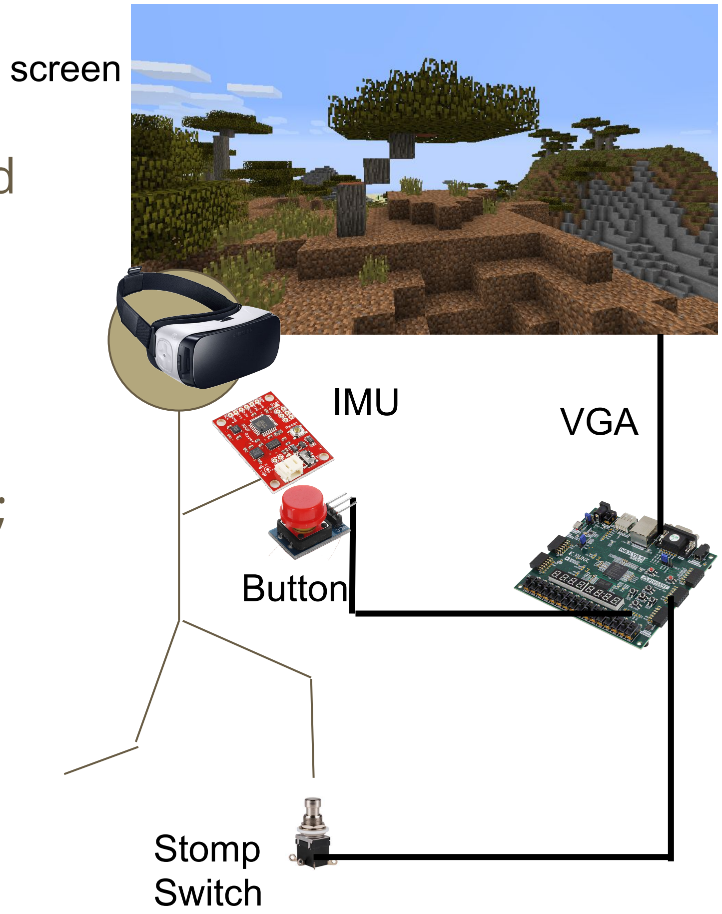
- Minecraft: a building game made of **blocks**.
- 3D, first person view.
- Navigate, mine, store and move blocks.

- **Hardware(simplified)?**

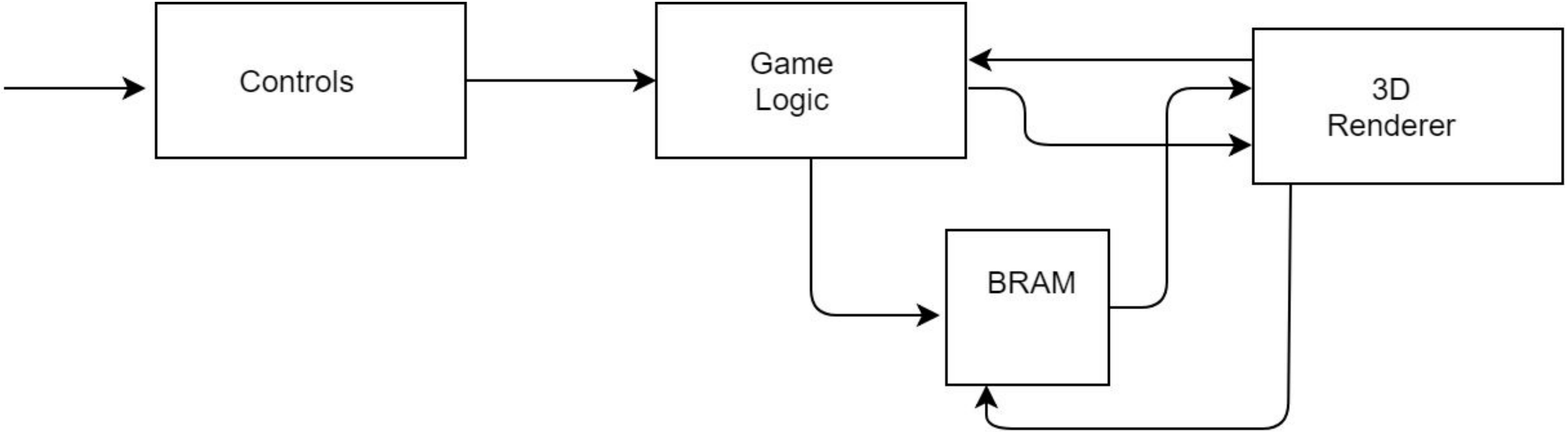
- **Interactive?**

Interactive Controls

- One **IMU chip** and one **button** attached to one hand (or one button on a hand and IMU on the head), and one **stomp switch** to step on.
- Control:
  - hand or head orientation: viewport;
  - Button: switch to interacting with blocks
  - Stomp switch: move or stay still. (Backward? Switch on FPGA)
- All wired. (If time allows: Bluetooth)

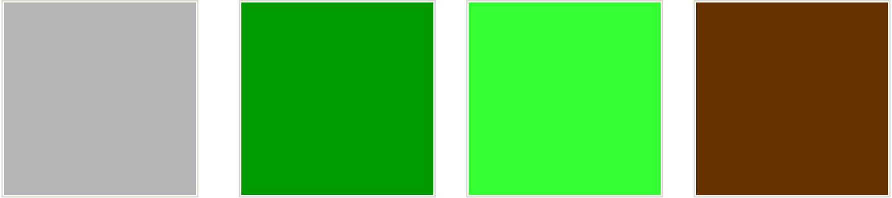


Very High Level Simplification



The world

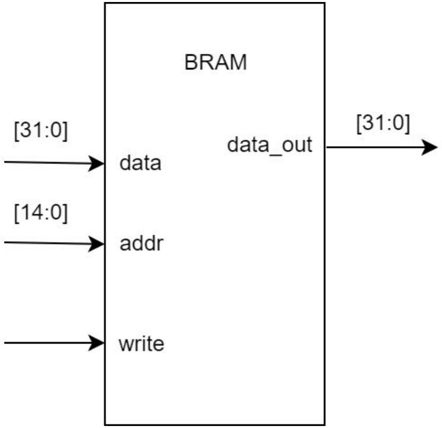
- A three dimensional world consisting of pre-generated 32x32x32 space.
- XYZ Coordinates, limited to positive direction.
- We will have stone, grass, leafs, and wood. All represented by a RGB value.
- Blue color when no block rendered (the sky).
- Positions represented by a 8 bit number, allowing for discrete movement within blocks (256/32 = 8).



R: 160 G: 160 B: 160	R: 0 G: 153 B: 0	R: 0 G: 255 B: 0	R: 102 G: 51 B: 0

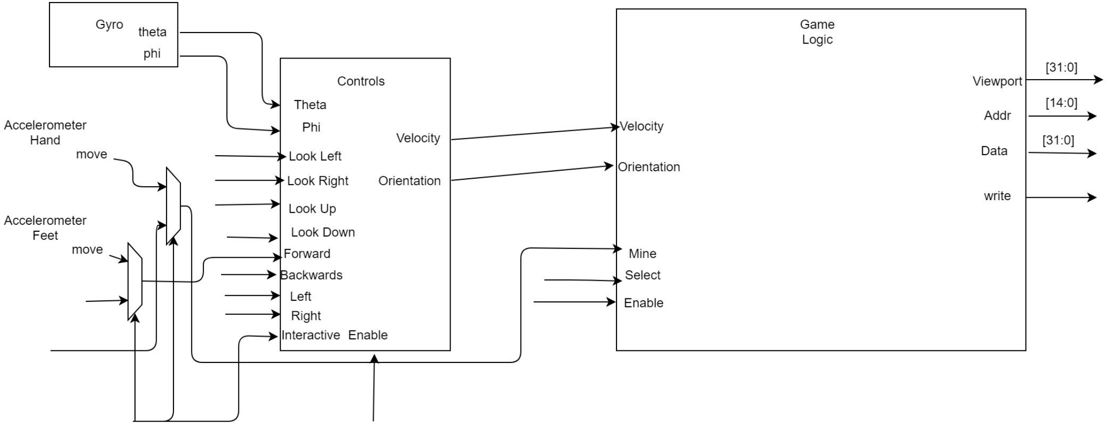
World state

- We represent Minecraft as a finite state machine.
- Suffices to make each block's position is a 5 bit number (integer positions)
- We save memory by representing the 4 colors as 2 bit numbers.
- Each block in space is either there or not (opaque or transparent)
- Together they form a state, which then the 3D player and 3D renderer react uniquely to.



Game Logic

- Responsible for updating the player’s position, updating viewport information and updating cube information.
- Inputs from controls tell what the game logic to do (ie: mine, look around, move).
- Updates the game state.



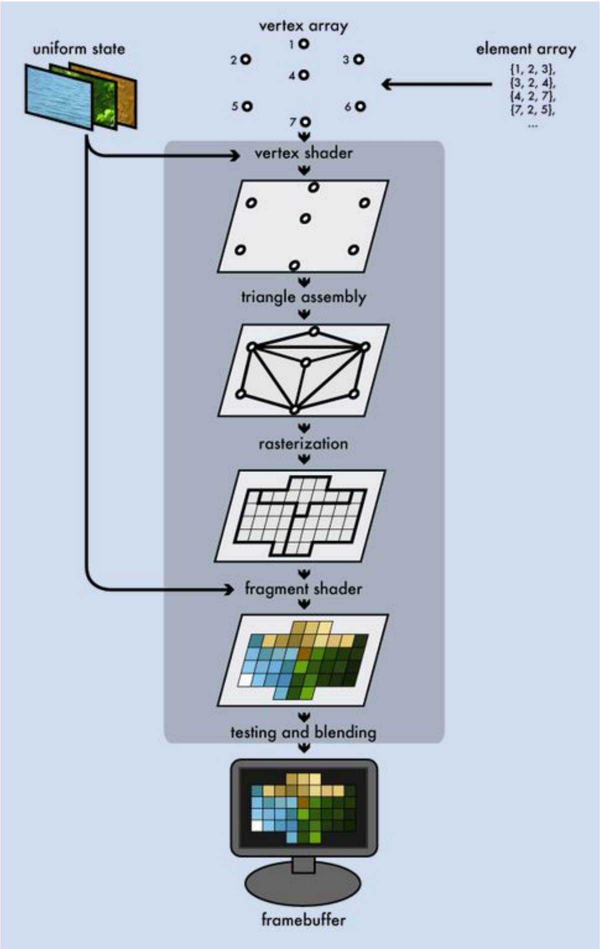
3D Rendering pipeline

- Start with an array of 3D coordinates, grouped 3 by 3 to form triangles. (Convenient for our cube world)

- Transform to 2D screen coordinates(also with depth information), clipping.

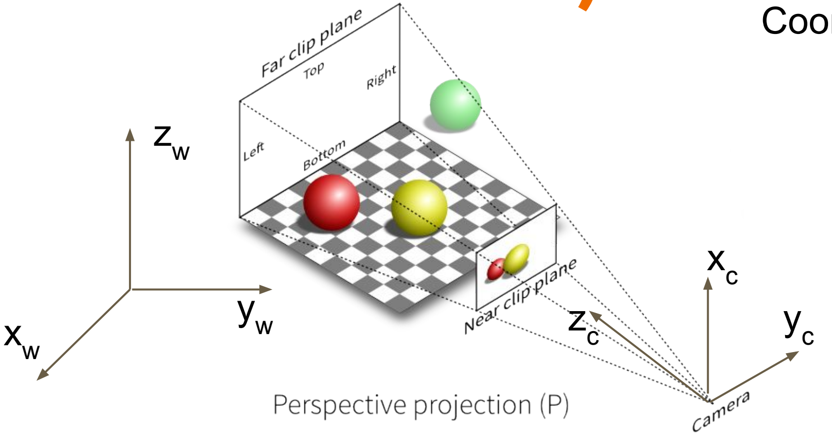
- Rasterization (based on three colors on the corners, fill in between.)

- Lighting and Shading (Not planned for this project.)



Credits: Joe Groff's blog

Coordinate Transformation (3d to 2d, with depth and orientation)



- **Homogeneous Coordinates**
- **Matrix multiplication** using **perspective projection**.
- Also store **Depth information(Zc)** and **normal direction**.
- **In parallel**.

Homogeneous Coordinate

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Translation

$$T = \begin{bmatrix} 1 & 0 & 0 & -x_c \\ 0 & 1 & 0 & -y_c \\ 0 & 0 & 1 & -z_c \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x - x_c \\ y - y_c \\ z - z_c \\ 1 \end{bmatrix}$$

Rotation

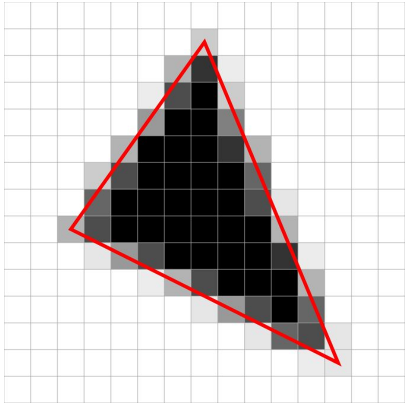
$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Perspective Projection

$$\begin{bmatrix} x_s \\ y_s \end{bmatrix} = \begin{bmatrix} d & 0 \\ 0 & d \end{bmatrix} \begin{bmatrix} \frac{x_w}{z_w} \\ \frac{y_w}{z_w} \end{bmatrix}$$

Rasterization

- Once 2D coordinates are given, we need to “fill” in the triangles. This is done by fulfilling tests for 3 linear equations.
- Which one? There could be conflicts?



Credit: wikimedia.org

$$E(x, y) = ax + by + c$$
$$E(x, y) \geq 0$$

Z-Buffering

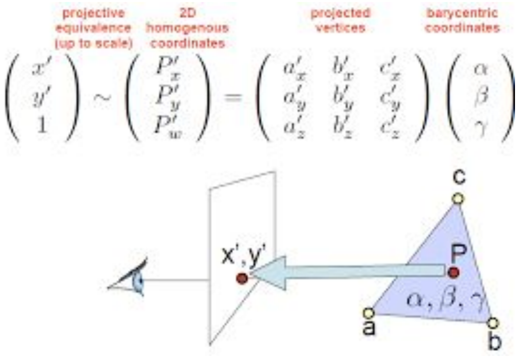
- Solution: Use interpolation to perform a depth test (known as z-buffering) to render the most closest pixel (save closest to the frame buffer).
- How? Interpolation using data from vertices. Use Barycentric.

For each triangle:

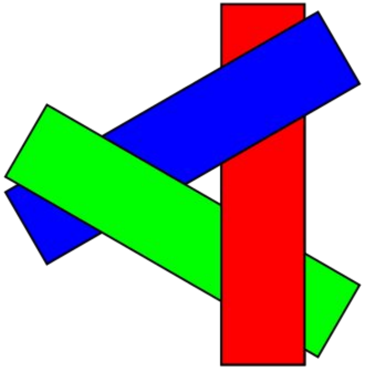
  For each pixel:

    Do rasterization

    Do Z-Buffering



Credits: MIT OCW Computer Graphics

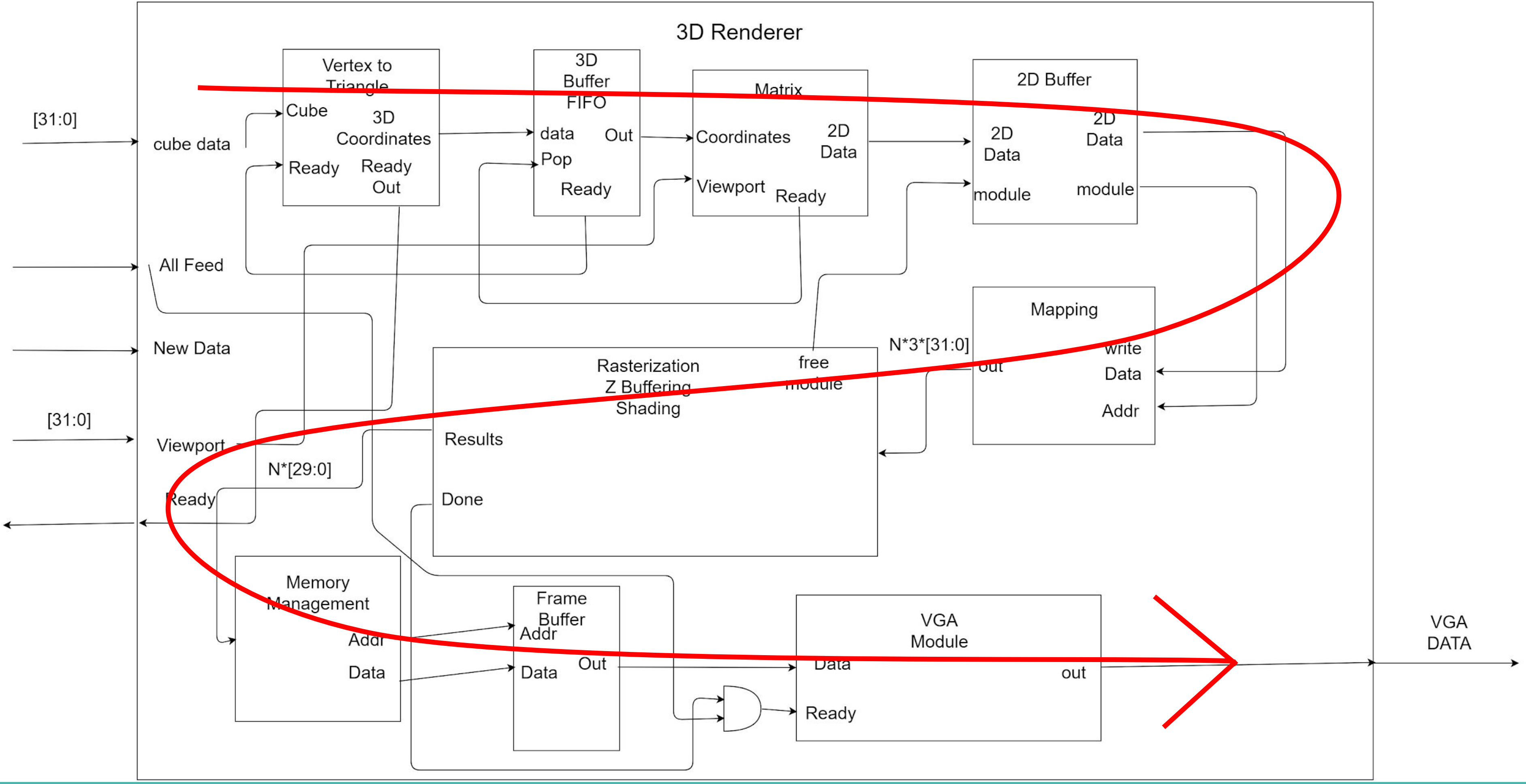


Credit: wikipedia.org

Parallelism in computing

- Many of these RB modules, each fed a triangle when empty.
- Similarly, many matrix multiplication modules.
- Computations can be independent of one another.
- These write to the frame buffer depending if they are the closest triangle
- Problems?
- Multiple writes in a single clock cycle.
- Solution: A memory management system.

3D Rendering block diagram



Priority

- The 3D renderer is the most complicated of the three main modules. It is also the most viable product. We do this first.
- Second is the Game Logic, we can play the game with FPGA buttons.
- Interactive controls module can be finished last.

Timeline

- 11/11: Finish module **matrix multiplication** and **rasterization modules** and testing, optimizing. First write in **C** or some high level language (to ensure it works).
- 11/18: Finish **buffering** and **simple game logic** (control, navigation). Display simple setups like **plain world** (maybe start with **static** world).
- 11/25: Start **incorporating hardware**, being able to **transfer data**. Improve game logic design, e.g. **adding interactions** like mining and placing objects, adding user inventories.
- 12/02: Finish controller design, namely add necessary **filters** to the data stream, **incorporate** it into the game logic. If have time add some **UI.**
- 12/09: **Debugging and optimization**. If have time, potential things to do: add some sound effects; add shading and texture for the blocks; add tools to use(sword, axe); add other creatures in the world.

Questions?